1 Machine Learning Engineer Nanodegree

1.1 Capstone Project

By Eakalak Suthampan
December 12th, 2017

2 WSDM - KKBox's Churn Prediction Challenge

Can you predict when subscribers will churn?

3 Definition

3.1 Project Overview

Customer churn prediction is essential for businesses. Early prediction of customers churn can help businesses to propose marketing campaign to prevent their customers from canceling their subscription, product or service. There are two broad approaches for churn analysis. Machine learning which we will focus and survival analysis. Machine learning methods, specifically classification, are widely used due to their high performance and ability to handle complex relationships in data. On the other hand, survival analyses can provide value by answering a different set of questions. Quantities, such as survival and hazard functions, can be used to forecast which customers will churn in a particular time period.

This capstone will take the problem from the Kaggle competition "WSDM - KKBox's Churn Prediction Challenge". The KKBOX is Asia's leading music streaming service and would like to build an algorithm that predicts whether a user will churn after their subscription expires. Currently, the company uses survival analysis techniques to determine the residual membership life time for each subscriber. By adopting different methods, KKBOX anticipates they'll discover new insights to why users leave so they can be proactive in keeping users dancing.

Some interesting practical works about churn prediction. Analyzing Customer Churn – Basic Survival Analysis demonstrates to use R Code to plot survival curve which shows relationship between days since subscribing and percent surviving. Analyzing Customer Churn by using Azure Machine Learning shows machine learning approach using Azure for customer churn prediction. It used ensemble of many classifiers such as Logistic Regression, Boosted Tree, Averaged Perceptron, SVM to predict customer churn.

3.2 Problem Statement

The objective for the Kaggle competition "WSDM - KKBox's Churn Prediction Challenge" is predicting whether a user will churn after their subscription expires. **The criteria of "churn" is no new valid service subscription within 30 days after the current membership expires.** I will build binary classification model to predict whether a user will 'churn' or 'not churn' after their subscription expires.

3.3 Metrics

Since we would like to identify customer churn as much as possible (high recall) while still getting acceptable precision. Therefore, I will use f1_score as evaluation metric since the f1_score conveys

the balance between the recall and the precision.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

$$precision = \frac{tp}{tp + fp}$$

$$recall = \frac{tp}{tp + fn}$$

By the way, the competition use binary logloss as evaluation metric. logloss is more precise to measure classification performance because it is based on prediction in probabilities. I think logloss is more appropriate when using in the competition but I would like to use f1_score because I think it is more intuitive to explain to business in term of recall, precision. The binary logloss equation is

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(p_i) + (1 - y_i)\log(1 - p_i))$$

where N is the number of observations, log is the natural logarithm, y_i is the binary target, and p_i is the predicted probability that y_i equals 1.

4 Analysis

4.1 Data Exploration

4.1.1 Datasets

There are datasets as follows.

1. train.csv (44.5 MB, 992931 rows) is the train set. The train data consists of users whose subscription expires within the month of February 2017.
    - msno: user id
    - is_churn: This is the target variable. Churn is defined as whether the user did not continue the subscription within 30 days of expiration. is_churn = 1 means churn,is_churn = 0 means renewal.
2. sample_submission_zero.csv (43.5 MB, 970960 rows) is the test set. The test data is with users whose subscription expires within the month of March 2017.
    - msno: user id
    - is_churn: This is what you will predict. Churn is defined as whether the user did not continue the subscription within 30 days of expiration. is_churn = 1 means churn,is_churn = 0 means renewal.
3. transactions.csv (1.61 GB, 21547746 rows) is transactions of users up until 2/28/2017.
    - msno: user id
    - payment_method_id: payment method
    - payment_plan_days: length of membership plan in days

- plan_list_price: in New Taiwan Dollar (NTD)
- actual_amount_paid: in New Taiwan Dollar (NTD)
- is_auto_renew
- transaction_date: format %Y%m%d
- membership_expire_date: format %Y%m%d
- is_cancel: whether or not the user canceled the membership in this transaction.

4. user_logs.csv (28.4 GB, 392106543 rows) describing listening behaviors of a user. Data collected until 2/28/2017.

    - msno: user id
    - date: format %Y%m%d
    - num_25: # of songs played less than 25% of the song length
    - num_50: # of songs played between 25% to 50% of the song length
    - num_75: # of songs played between 50% to 75% of of the song length
    - num_985: # of songs played between 75% to 98.5% of the song length
    - num_100: # of songs played over 98.5% of the song length
    - num_unq: # of unique songs played
    - total_secs: total seconds played

5. members.csv (352 MB, 5116194 rows) consist of user information. Note that not every user in the dataset is available.

    - msno
    - city
    - bd: age. Note: this column has outlier values ranging from -7000 to 2015, please use your judgement.
    - gender: this column has NULL values
    - registered_via: registration method
    - registration_init_time: format %Y%m%d
    - expiration_date: format %Y%m%d, taken as a snapshot at which the member.csv is extracted. Not representing the actual churn behavior.

There are additional datasets that were added later because the version 1 testing dataset (sample_submission_zero.csv) were leaked. So Kaggle provided the new version 2 test dataset and more training sets as folows.

6. sample_submission_v2.csv (40.6 MB) is the new test dataset for April, 2017. (replaced sample_submission_zero.csv)
7. train_v2.csv (43.5 MB) contains additional churn data for March, 2017.
8. transactions_v2.csv (110 MB) contains additional transactions data until 3/31/2017.
9. user_logs_v2.csv (1.33 GB) contains additional user logs data until 3/31/2017.
10. members_v3.csv (408 MB) is the new member dataset. (replaced members.csv)

for more information about the datasets please see at the Kaggle competition "WSDM - KKBox's Churn Prediction Challenge".

4.1.2 Feature Engineering

I managed to create the following new features from transactions.csv.

- last transaction for each user: I guess that features such as payment_method_id, payment_plan_days, is_auto_renew, membership_expire_date, is_cancel may effect the churn.
- subscribe duration: If users have been subscribed for a long time, they should have less chance to churn.
- oldest transaction date for each user: Some users may have very oldest transaction date but a short subscribe duration.
- count of transactions for each user: may be the more transactions, the less likely to churn.

I managed to create the following new features from user_logs.csv.

- aggregate values such as total seconds played, total songs played for each user may effect the churn.
- count of all logs for each user. This can tell number of activities for a user.
- count of logs for the last 1 month for each user. If users don't have much activities in the last month (small log counts) they might have more chance to churn.

Finally, I merge all above features to the members.csv. Informations in the members.csv such as user age and registration method may effect the churn. For all merging, I use left join to merge features so there are NAN values for records that cannot be merged.

4.1.3 Training dataset information

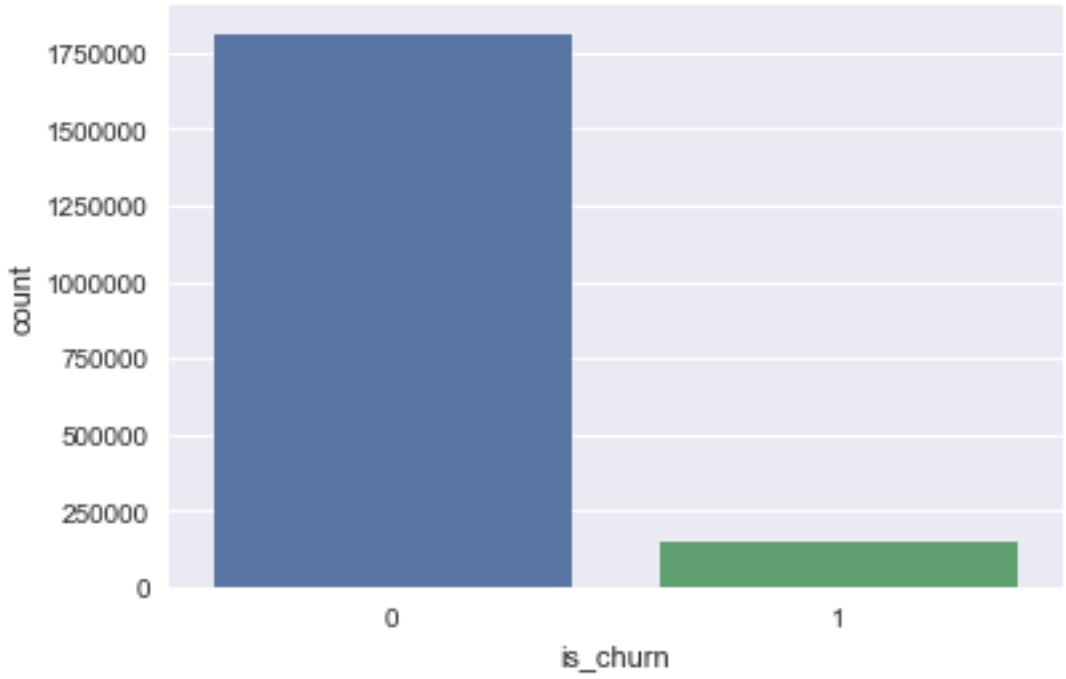
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1963891 entries, 0 to 1963890
Data columns (total 27 columns):
msno                        object
is_churn                    int64
payment_method_id           int64
payment_plan_days           int64
plan_list_price             int64
actual_amount_paid          int64
is_auto_renew               int64
transaction_date            int64
membership_expire_date      int64
is_cancel                   int64
subscribe_days              float64
oldest_transaction_date     int64
transaction_count           float64
num_25                      float64
num_50                      float64
num_75                      float64
num_985                     float64
num_100                     float64
num_unq                     float64
total_secs                  float64
```

```
log_count_all              float64
log_count_1_month          float64
city                       float64
bd                         float64
gender                     object
registered_via             float64
registration_init_time     float64
dtypes: float64(15), int64(10), object(2)
memory usage: 404.5+ MB
None
```

4.2 Exploratory Visualization

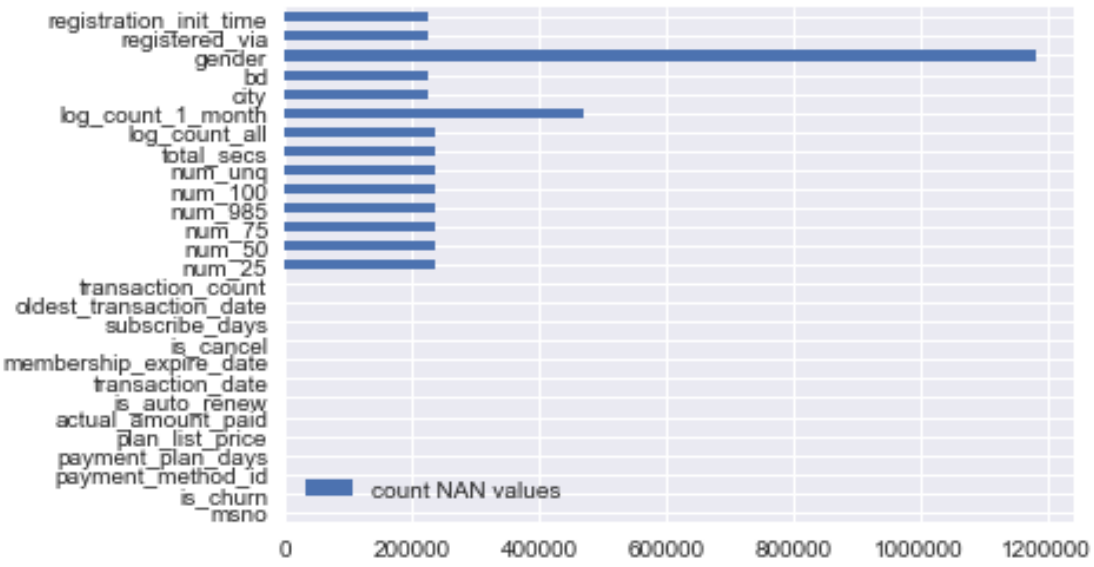
4.2.1 Labels Count

First, let see the counts of labels. You can see that the labels are very imbalanced. The count of 'is_churn'=1 (churn) is very less than 'is_churn'=0 (not churn). The prediction model may be biasd to the majority which is not churn.



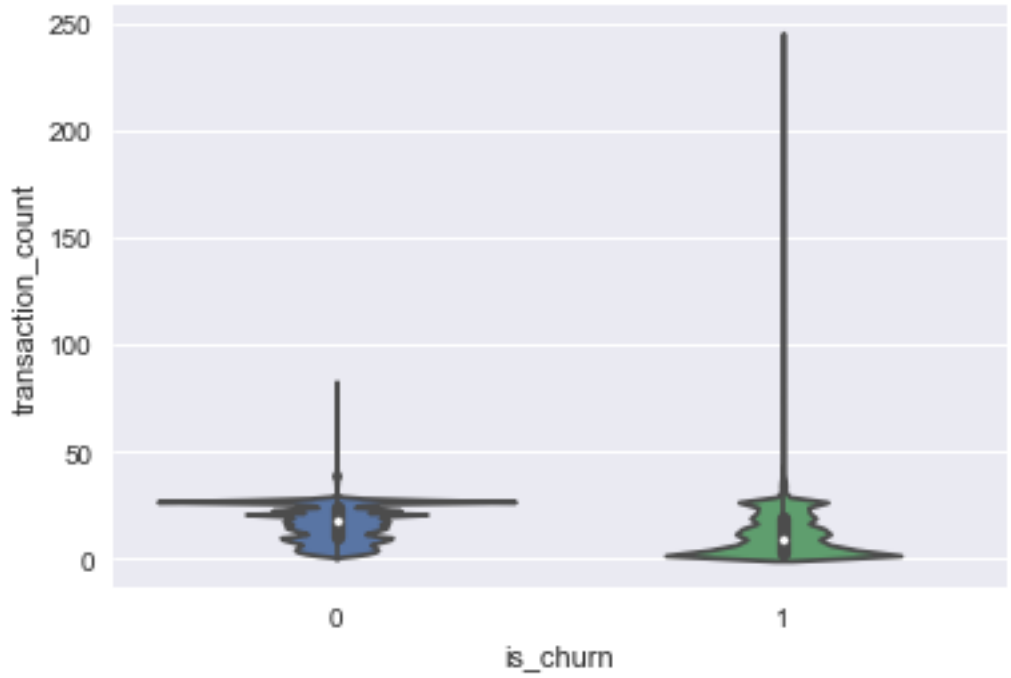
4.2.2 Count of NAN Values

There are NAN values for the merged train dataset. The gender column has NAN as majority value. The other columns have NAN values because I used left join and some users don't have associate data in users_log.csv and members.csv.



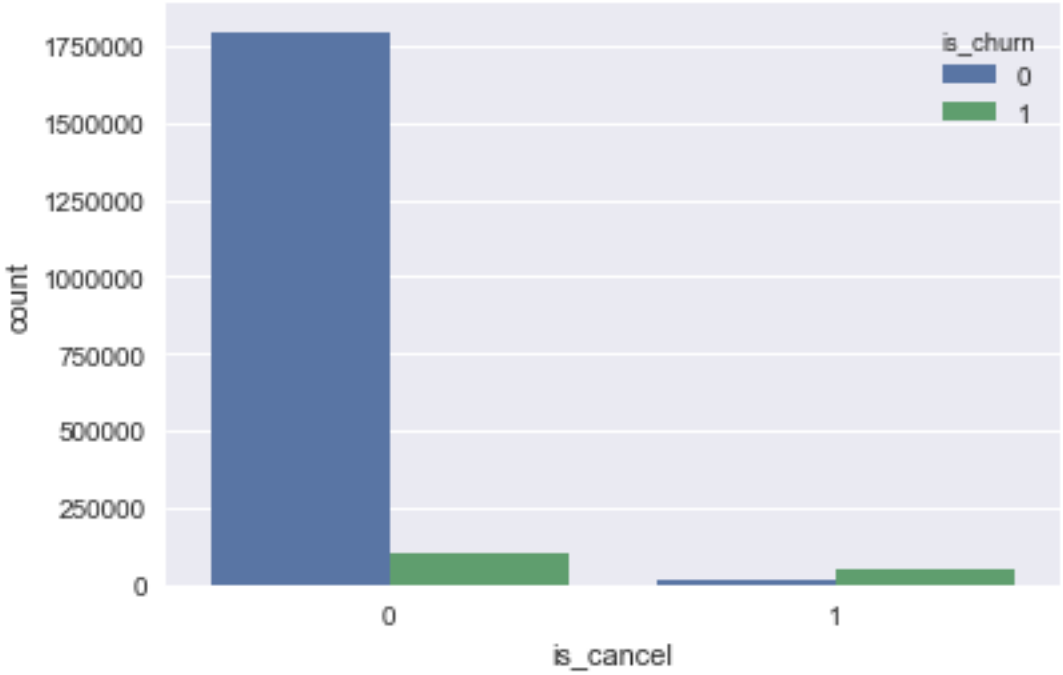
4.2.3 Count of transactions

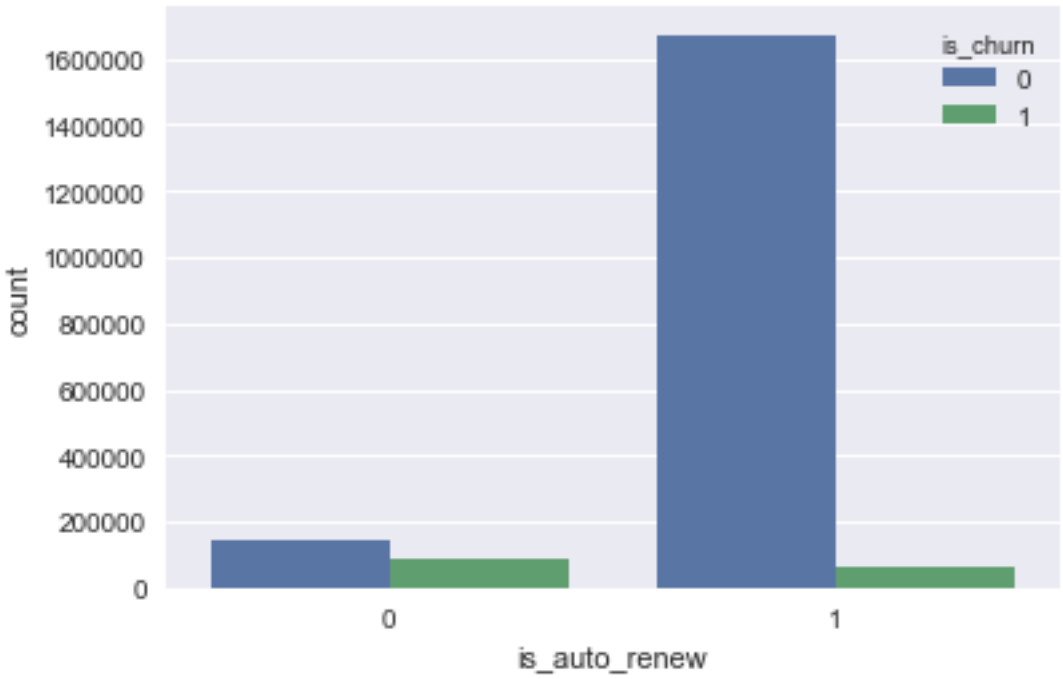
Vilolin plot for the count of transactions for each user shows that lower transaction counts have higher probabilities to churn (is_churn=1) while higher transaction counts have higher probabilities to not churn (is_churn=0). There are many outliers for the 'is_churn=1' (long tail).



4.2.4 is_auto_renew, is_cancel

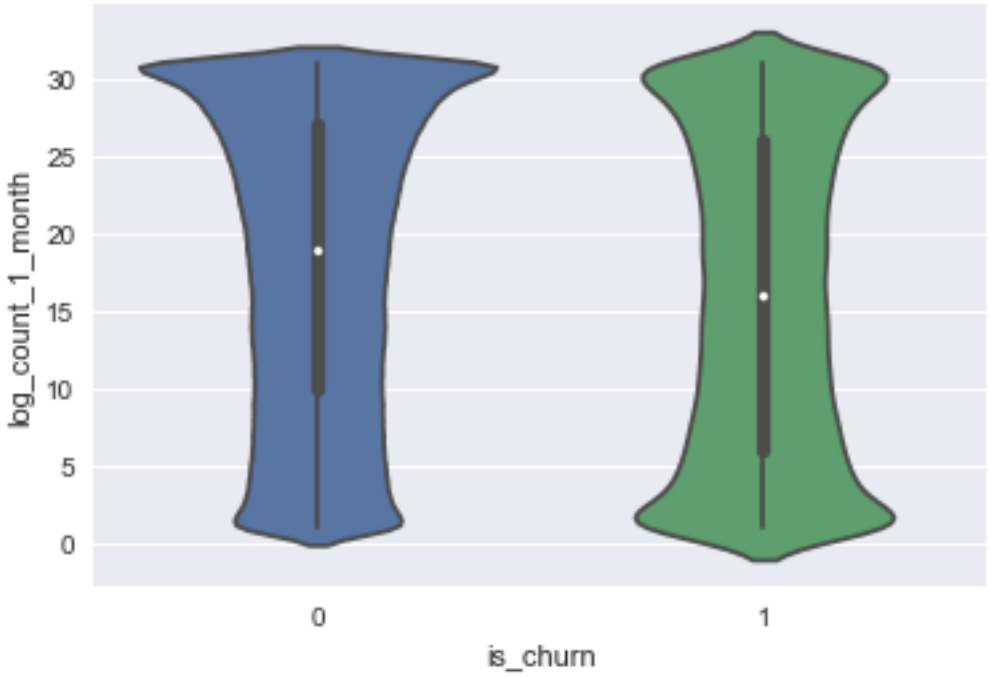
Show count plot for 'is_auto_renew' and 'is_cancel'. The plots show that 'is_cancel=1' has high effect for user to churn while 'is_auto_renew=1' has very high effect for user to not churn.





4.2.5 Count of last month logs

The count of last month logs means how much activities for user in the last month. From the plot, 'is_churn=1' has higher occurrence at both the top and bottom of the plot while 'is_churn=0' has very high occurrence at the top of the plot.



4.3 Algorithms and Techniques

First, before discussing about algorithms let's summarize the characteristics of the train data.

- very imbalanced label (churn data is very small compared to not churn data).
- have a lot of missing values.
- many columns have outlier values.
- all columns are numeric or make sense to be numeric (for example, transaction_date, membership_expire_date).

From the characteristics of the dataset, I think ensemble method such as Random Forest and XGBoost are the most appropriate.

- Random Forest uses bagging while XGBoost uses boosting to handle the imbalanced classification.

- They don't need to do preprocessing steps like scaling and standardize.
- They can handle the missing values.
- XGBoost may be better in term of accuracy but need more parameter tuning than Random Forest.
- Random Forest may be better in term of less overfitting if there are outlier values.
- Randome Forest can use both numeric and category data while XGBoost can use only numeric data.

Random Forest is a bagging ensemble method working by training multiple decision tree models on different samples (with replacement) and different random features then average their predictions.

$$\hat{y} = \frac{1}{N}\sum_{k=1}^{N} f_k(x_i)$$

where f_k is a random features decision tree classifier and x_i is a random samples with replacement from training data and N is number of decision tree classifier. It aim to decrease variance not bias. This lecture describes how it can reduce variance without changing bias.

Assume we measure a random variable x with a $N(\mu, \sigma)$ distribution. If random variable x is measured k times $(x_1, x_2, ...x_k)$ and the value is estimated as: $\frac{1}{k}(x_1 + x_2 + \ldots + x_k)$.

- Mean of the estimate is still μ
- But, variance is smaller $\frac{1}{k^2}(Var(x_1) + \ldots Var(x_k)) = \frac{k\sigma^2}{k^2} = \frac{\sigma^2}{k}$

XGBoost is a boosting ensemble method working by starting with a very simple decision tree model (accuracy just slightly better than 50%) then incrementally training each new decision tree model instance to emphasize the training instances that previous model misclassified then use majority voting of these boosting models. Equation for additive training (fix what we have learned, and add one new tree at a time) is

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

where $\hat{y}_i^{(t)}$ is a prediction value at step t and f_t is a new decision tree at step t and x_i is samples from the distribution that weighted the misclassified instances from the previous step. XGBoost aim to decrease bias, not variance.

4.4 Benchmark

I will use two simple benchmarks as follows

- use Dicision Tree as simple model benchmark to be compared with the Random Forest and XGBoost.
- use a simple rule, if a user has no activities in the last 30 days then predict the user as churn.

5 Methodology

5.1 Data Preprocessing

I will do the following data preprocessing.

- convert column gender from category to be numeric (1=male, 2=female) because XGBoost need all features to be numeric.
- drop msno column which is user id (does not need user id when training model).
- Most of NAN values occured because using left join so I think it make sense to fill NAN with 0.

XGBoost and Random Forest don't need other feature transformations such as scaling or standardizing.

5.2 Implementation

I will do following for tuning the initial models.

- split 10% of train data to be the test data. This will be used for comparing models.
- train and compare two models, Random Forest and XGBoost.
- tune model parameters using GridSearchCV (using small samples and small n_estimates for training speed).

5.2.1 XGBoost

I will fixing the following parameters for GridSearchCV.

- n_estimators = 50 (using large n_estimators will take very long time to run)
- using 100000 records of the train data. (about 5% of train data)
- cv = 4 (cross validate)
- scoring = 'log_loss' (since Kaggle use log_loss as evaluation metric)
- objective='binary:logistic' (binary classification)

I will tune the following parameters using GridSearchCV.

- "max_depth": Maximum tree depth for base learners.
- "min_child_weight": Minimum sum of instance weight(hessian) needed in a child.
- "subsample": Subsample ratio of the training instance.
- "colsample_bytree": Subsample ratio of columns when constructing each tree.

the parameters range to tune are as follows.

- "max_depth": [4,5,6,7]
- "min_child_weight" :[1,2,3]
- "subsample": [0.6,1.0]
- "colsample_bytree": (0.6,1.0)}

The best parameters from GridSearchCV are

```
Out[22]: {'colsample_bytree': 1.0,
          'max_depth': 6,
          'min_child_weight': 1,
          'subsample': 0.6}
```

The first tuning scores on the train data are

```
log_loss = 0.0744267875661, recall = 0.757172395968, f1_score = 0.809142383649
```

The first tuning scores on the test data are

```
log_loss = 0.0793903277045, recall = 0.71131547381, f1_score = 0.766039902397
```

5.2.2 Random Forest

I will fixing the following parameters for GridSearchCV.

- n_estimators = 50 (using large n_estimators will take very long time to run)
- using 100000 records of the train data (about 5% of train data).
- cv = 4 (cross validate)
- scoring = 'log_loss' (since Kaggle use log_loss as evaluation metric)

I will tune the following parameters using GridSearchCV.

- max_features: The number of features to consider when looking for the best split.
- max_depth: The maximum depth of the tree.
- min_samples_leaf: The minimum number of samples required to be at a leaf node.
- min_samples_split: The minimum number of samples required to split an internal node.

the parameters range to tune are as follows.

- "max_depth": [4,5,6]
- "min_samples_split" :[2,3,4]
- "min_samples_leaf": [1,3,5]
- "max_features": (4,5,6)

I intent to tune Random Forest to less overfit than XGBoost to compare results when submiting to Kaggle.

The best parameters from GridSearchCV are

```
Out[26]: {'max_depth': 6,
          'max_features': 6,
          'min_samples_leaf': 1,
          'min_samples_split': 4}
```

The first scores on the train data are

```
log_loss = 0.0867531737872, recall = 0.719048849832, f1_score = 0.777692361451
```

The first scores on the test data are

```
log_loss = 0.0879615918596, recall = 0.703851792616, f1_score = 0.762709416522
```

Complications during the implementation are

- GridSearchCV takes very long time to brute force searching the best parameters. So I need to use just small samples of the training data. The parameter 'n_estimator' also has a huge effect to the running time of GridSearchCV.
- There are so many parameters for the algorithms but I can not fine tune them because adding one more choice of the parameter can add multiplication in the time consuming so it's hard to decide which parameters and how much of its range need to be tuned. May be I should consider RandomSearchCV instead of GridSearchCV for the next time.
- For tuning Random Forest, increasing values for 'max_depth' and 'max_features' will get better score on the test data but I've found that it will overfit when submiting results to Kaggle.

5.3 Refinement

For final tuning, I will use parameters from the previous GridSearchCV and do following to tune the final scores.

- increase n_estimates to be 500 for both XGBoost and Random Forest.
- train on the full train data.
- use the same test data as previous phase.

5.3.1 XGBoost

The final scores on the test data are

```
log_loss = 0.0747653657696, recall = 0.726042916167, f1_score = 0.770590939633
```

5.3.2 Random Forest

The final scores on the test data are

```
log_loss = 0.0878255092431, recall = 0.711781953885, f1_score = 0.764785908635
```

5.3.3 Scores Summary

Compare scores for intermediate tuning on both train and test data and final tuning on test data.
for XGBoost

metrics	first_score_on_train_data	first_score_on_test_data	final_score_on_test_data
log_loss	0.07442678756606426	0.07939032770447087	0.07476536576963773
recall	0.7571723959679504	0.7113154738104758	0.7260429161668666
f1_score	0.8091423836486672	0.7660399023970144	0.770590939632917

for Random Forest

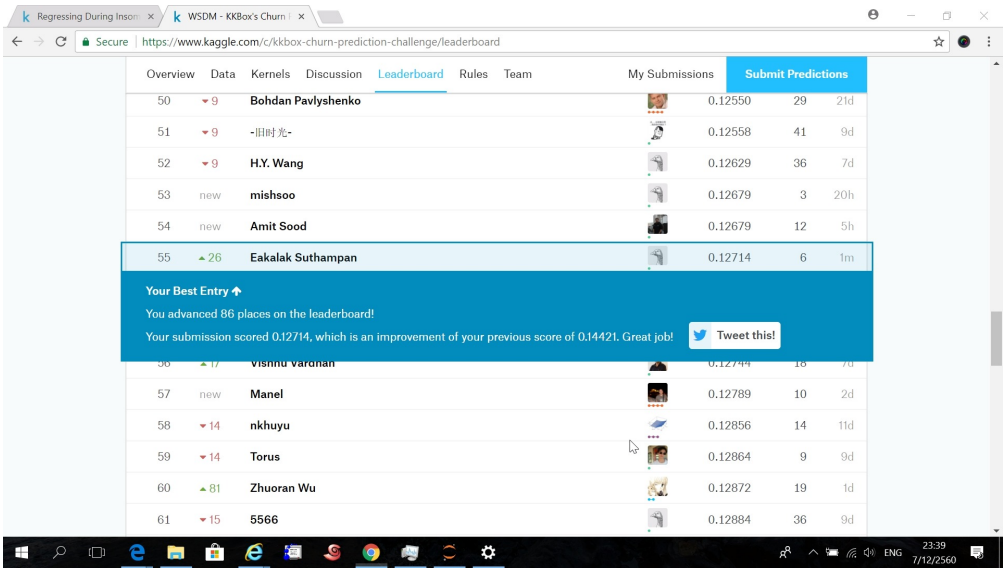
metrics	first_score_on_train_data	first_score_on_test_data	final_score_on_test_data
log_loss	0.08675317378719496	0.08796159185956462	0.08782550924311255
recall	0.7190488498319979	0.7038517926162868	0.7117819538851127
f1_score	0.7776923614508351	0.7627094165222416	0.7647859086352571

For both XGBoost and Random Forest, the scores on both train and test data are very close and seem not overfit. XGBoost final results on the test data have better scores than Random Forest.

6 Results

6.1 Model Evaluation and Validation

I submitted results from both XGBoost and Random Forest to Kaggle. The best result is from Random Forest model which I got 55th place out of 434 (top 13%) on the Kaggle public leaderboard (As of DEC 7, 2017).



Model	log_loss_on_Kaggle_public_leaderboard	log_loss_on_the_test_data
XGBoost	0.14421	0.07476536576963773

Model	log_loss_on_Kaggle_public_leaderboard	log_loss_on_the_test_data
Random Forest	0.12714	0.08782550924311255

It's surprised me that Random Forest got higher score on Kaggle Public Leaderboard although XGBoost got higher score on the test data. It seems that both models are overfit to the real data but Random Forest is less overfit than XGBoost. I think the reason that Random Forest is less overfit is because both model have the parameter 'maxdepth' equally which is 6 so Random Forest which is bagging seem to less overfit than XGBoost which is boosting.

So my final chosen model is Random Forest because in real life many of user churn may don't have exact pattern so the less overfit model should be better.

6.2 Justification

I have the following benchmarks.

- benchmark1: use a simple decision tree.
- benchmark2: use a simple rule if user does not have usage log in the last month then predict user to churn.

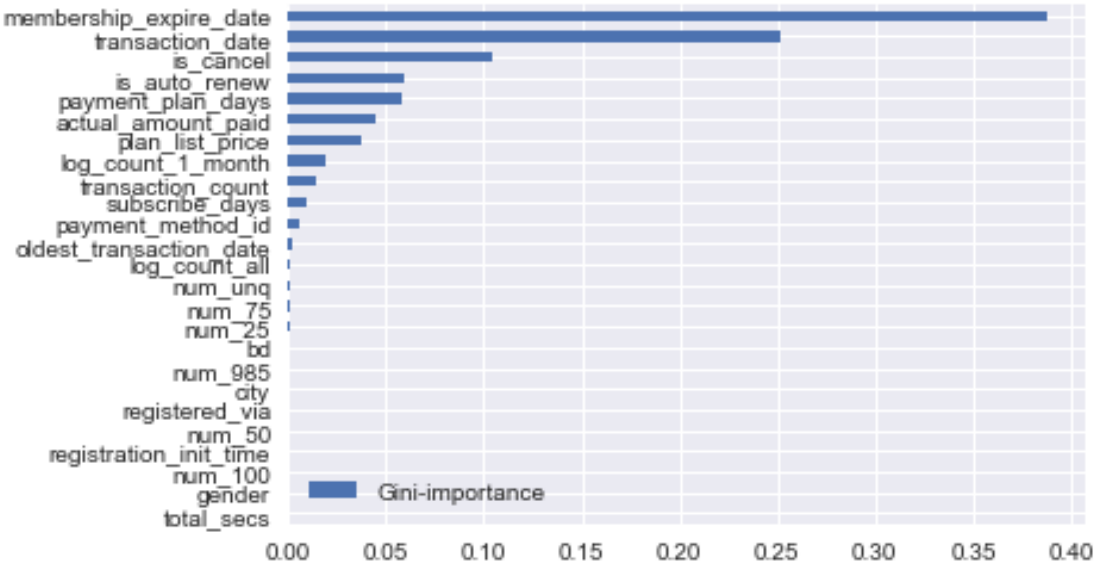
Here are results of bechmarks to compare with the final chosen model.

model	log_loss	recall	f1_score
benchmark1	1.72517900298467	0.6258829801412769	0.6398909896099473
benchmark2	8.747856170892863	0.413567906170865	0.1997039516025228
Random Forest	0.08782550924311255	0.7117819538851127	0.7647859086352571

- benchmark1 and benchmark2 cannot predict in probabilities so their log_loss are very high.
- benchmark1 has both recall and f1_core significantly lower than Random Forest.
- benchmark2 has both recall and f1_core significantly lower than Random Forest. Especially the very low f1_score compared to recall tells that it has very low precision (high False Positive).
- Random Forest has high enough scores on both recall and f1_score which means it can effectively identify around 71 out of 100 for the churn users while still maintain acceptable false alarm.

7 Conclusion

7.1 Free-Form Visualization



Feature importances plot shows that 'membership_expire_date', 'transaction_date' and 'is_cancel' are the top 3 most important features to predict customer churn. Most of important features are from transactions.csv. Features from user_log.csv and members.csv are almost no importance.

7.2 Reflection

This project is an imbalanced binary classification problem. Kaggle provides so many large datasets and so many features and I need to merge these datasets together to derive some new useful features. This feature engineering step is very time consuming for me and it consumed almost all of 32GB of my RAM. After that I need to do data exploration and visualization to understand more on the data. In the preprocessing step, I need to clean the NAN values by just simply filling them with zero and drop some columns that not meaningful to train model.

I choose ensemble tree based model Random Forest and XGBoost as they are effective technique to handle imbalanced classification. For evaluation metrics, Kaggle uses logloss for this problem and I also use f1 along with recall as they can explain to business in term of how many user churn can be identified and how many false alarms.

For the train phase, I trained two model XGBoost and Random Forest. I use GridSearchCV to fine tune the best parameters. This phase is also a very time comsuming since GridSearchCV is a brute force method and the training data are quite large. The scores on the train and test data are quite similar and seem not overfit. XGBoost is better. It got around 0.07-0.08 for the logloss on the test data while Random Forest got around 0.08-0.09. These scores are very much far better than the simple rule based benchmark (if user has no activity longer than 30 days then predict as churn) and non ensemble method benchmark like decision tree.

I submitted results from both XGBoost and Random Forest to Kaggle and it's surprised me that Random Forest got very much better scores on the Kaggle Public Learderboard although XGBoost

score is better on my test data. I got around 55th place out of 434 (top 13%) for the Random Forest as the day I submitted. Random Forest logloss score on the Kaggle Public Learderboard is around 0.12 while XGBoost is around 0.14. Both models seem to be overfit when compare to the score on my test data.

7.3 Improvement

To improve the model I will focus on reducing overfit.

- Explore and filter out more outlier values.
- Remove unnecessary features based on the feature importances plot.
- Tuning parameters that effect the overfit such as reducing maxdepth.
- Try early stoping technique in XGBoost which can help preventing overfit.
- Train with more data if possible to make model more generalize.
- Try to create new feature that does not depend on date value and not strong correlate with the existing features.
- Ensemble result of both XGBoost and Random Forest.